TASK1:- what is the difference between procedural and declarative language and imperative ?

Declarative programming refers to code that is concerned with higher levels of abstraction.

Imperative programming refers to code that is concerned with lower levels of abstraction.

Procedural programming is a subset of imperative programming which utilizes subroutines

TASK2:- how does python deal with large numbers more than 14 bytes?

Python supports a "bignum" integer type which can work with arbitrarily large numbers. In Python 2.5+, this type is called long and is separate from the int type, but the interpreter will automatically use whichever is more appropriate. In Python 3.0+, the int type has been dropped completely.

That's just an implementation detail, though - as long as you have version 2.5 or better, just perform standard math operations and any number which exceeds the boundaries of 32- bit math will be automatically (and transparently) converted to a bignum.

TASK3:-what is the null pointer excpection

In computing, a **null pointer** or **null reference** is a value saved for indicating that the pointer or reference does not refer to a valid object. Programs routinely use null pointers to represent conditions such as the end of a list of unknown length or the failure to perform some action; this use of null pointers can be compared to nullable types and to the *Nothing* value in an option type.

A null pointer should not be confused with an uninitialized pointer: a null pointer is guaranteed to compare unequal to any pointer that points to a valid object. However, depending on the language and implementation, an uninitialized pointer may not have any such guarantee. It might compare equal to other, valid pointers; or it might compare equal to null pointers. It might do both at different times; or the comparison might be undefined behaviour.

TASK4:-case insensitive programming langues

- ABAP,
- Ada
- BASICs
- Fortran,
- SQL
- Pascal
- Haskell,
- Prolog
- Go.

TASK5:-what is the difference between heap and stack?

Parameter	Stack	Heap
Type of data structures	A stack is a linear data structure.	Heap is a hierarchical data structure.
Access speed	High-speed access	Slower compared to stack
Space management	Space managed efficiently by OS so memory will never become fragmented.	Heap Space not used as efficiently. Memory can become fragmented as blocks of memory first allocated and then freed.
Access	Local variables only	It allows you to access variables globally.
Limit of space size	Limit on stack size dependent on OS.	Does not have a specific limit on memory size.
Resize	Variables cannot be resized	Variables can be resized.
Memory Allocation	Memory is allocated in a contiguous block.	Memory is allocated in any random order.
Allocation and Deallocation	Automatically done by compiler instructions.	It is manually done by the programmer.
Deallocation	Does not require to de-allocate variables.	Explicit de-allocation is needed.
Cost	Less	More
Implementation	A stack can be implemented in 3 ways simple array based, using dynamic memory, and Linked list based.	Heap can be implemented using array and trees.
Main Issue	Shortage of memory	Memory fragmentation

Locality of reference	Automatic compile time instructions.	Adequate
Flexibility	Fixed size	Resizing is possible
Access time	Faster	Slower
Advantages	1-Helps you to manage the data in a Last In First Out(LIFO) method which is not possible with Linked list and array. 2-When a function is called the local variables are stored in a stack, and it is automatically destroyed once returned. 3-A stack is used when a variable is not used outside that function. 4-It allows you to control how memory is allocated and deallocated. Stack automatically cleans up the object. 5-Not easily corrupted Variables cannot be resized.	1-Heap helps you to find the greatest and minimum number 2-Garbage collection runs on the heap memory to free the memory used by the object. 3-Heap method also used in the Priority Queue. 4-It allows you to access variables globally. 5-Heap doesn’t have any limit on memory size.
Disadvantages	1-Stack memory is very limited. Creating too many objects on the stack can increase the risk of stack overflow. 2-Random access is not possible. 3-Variable storage will be overwritten, which sometimes leads to undefined behavior of the function or program. 4-The stack will fall outside of the memory area, which might lead to an abnormal termination.	1-It can provide the maximum memory an OS can provide 2-It takes more time to compute. 3-Memory management is more complicated in heap memory as it is used globally. 4-It takes too much time in execution compared to the stack.
When to use the Heap or stack?	You should use heap when you require to allocate a large block of memory. For example, you want to create a large size array or big structure to keep that variable around a long time then you should allocate it on the heap. However, If you are working with relatively small variables that are only required until the function using them is alive. Then you need to use the stack, which is faster and easier.	

TASK6:- What programming languages does auto garbage collection support ? and which do not?

Many programming languages require garbage collection, either as part of the language specification (e.g., RPL, Java, C#, D,[4] Go, and most scripting languages) or effectively for practical implementation (e.g., formal languages like lambda calculus). These are said to be garbage-collected languages. Other languages, such as C and C++, were designed for use with manual memory management, but have garbage-collected implementations available. Some languages, like Ada, Modula-3, and C++/CLI, allow both garbage collection and manual memory management to co-exist in the same application by using separate heaps for collected and manually managed objects. Still others, like D, are garbage-collected but allow the user to manually delete objects or even disable garbage collection entirely when speed is required .